The Fun of using TCP for an MMORPG

Carsten Griwodz[1,2], Pål Halvorsen[1,2]
[1]IFI, University of Oslo, Norway [2]Simula Research Laboratory, Norway
{griff, paalh}@ifi.uio.no

ABSTRACT

Massive multi-player online games have become a popular, fast growing, multi-million industry with a very high user mass supporting hundreds or thousands of concurrent players. In many cases, these games are centralized and every player communicates with the central server through a time-critical unicast event stream. Funcom's Anarchy Online is one of these; it is based on TCP. We find that its kind of traffic has some interesting properties that inspire changes to protocol or architecture.

In these game streams, TCP does not back off, using TCP does not have to be slower than using UDP, and almost only repeated timeouts ruin the game experience. Improving the latter in the sender implementation does not impose any remarkable penalty on the network. Alternatively, a proxy architecture for multiplexing could save about 40% resources at the server, allow congestion control to work and also reduce the lag of the game.

Keywords

MMORPGs, thin streams, TCP performance

1. INTRODUCTION

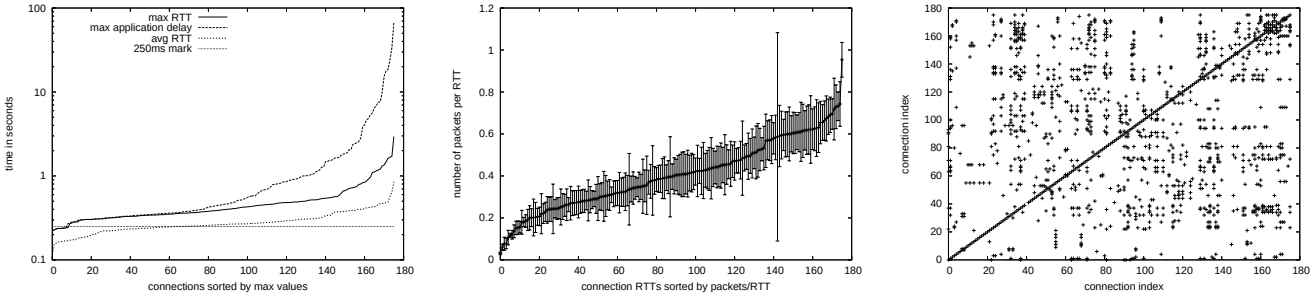
Large improvements in computer technology have enabled highly interactive distributed applications such as distributed virtual environments, massive multi-player online games (MMOGs) and computer supported cooperative work. These applications often include several types of media ranging from text to continuous media and may have stringent requirements with respect to the quality of the client data playout.

In the MiSMoSS project, we aim for better system support for such applications by trying to make more efficient use of the available resources and increase the perceived quality at the user. In particular, we look at MMOGs due to the mix of different media, the stringent latency requirements, the dynamic client groups and the fact that it has become a popular, fast growing, multi-million industry with a very high user mass. Today, MMOGs are increasing in size and complexity, supporting hundreds or thousands of concurrent players [13], and they typically include a mixture of game situations from role-playing games, first person shooter games and real-time strategy games. Players in the game move around and interact with other players, seemingly as if they were located next to each other. Frequently, many players interact with each other and the same object in the game world; these are then said to share an area-of-interest (AoI) or to be within each others AoI. In such a case, it is important that the users get the same information more or less at the same time to have a consistent view of the game world.

Today, most MMOGs apply a central server (or a central server cluster) approach for collecting and processing game events generated by players, and point-to-point communication for the distribution of game state updates. With respect to delivering events to players in time, the game traffic must compete with all other traffic in the network, and in case of errors, the servers usually use standard protocol implementations available in the operating system kernels.

In this paper, we focus on the challenge of delivering real-time data in MMOGs. As a starting point, we examine the traffic of a particular game, Funcom's popular massive multi-player online role playing game (MMORPG) *Anarchy Online* [10], which uses an unmodified default TCP variation to deliver data to the players. A trace from one of Anarchy Online's game regions shows that the event streams are very thin (i.e., less than 4 small packets per second), and some users experience huge delays due to retransmissions. We therefore look at the various TCP mechanisms in Linux to see if the experienced game play can be improved. Our experimental results show that, in such thin game streams, 1) TCP does not back off, i.e., each stream may be thin, but the application may have hundreds or thousands concurrent streams which together may give congestion without reducing the sending rate, 2) using TCP does not have to be slower than using UDP as the send buffer is usually empty and an event may be sent immediately, and 3) almost only repeated timeouts ruin the game experience because the number of in-flight packets is so small that fast retransmissions due to multiple duplicate acknowledgments is an exceptional occurance. Furthermore, improving the latter in the sender implementation does not impose any remarkable penalty on the network. Alternatively, a proxy architecture for multiplexing could save about 40% resources at the server, allow congestion control to work and also reduce the lag of the game.

NOSSDAV '06 Newport, Rhode Island USA



(a) RTT versus maximum application delay

(b) Packets per RTT with standard deviation

(c) Matrix of connections that probably share a path

Figure 1: Anarchy Online packet trace analysis

The rest of this paper is organized as follows. In section 2, we briefly describe some of the main observations from the Anarchy Online game trace. Section 3 presents some related work, and experiments and results using the various Linux implementations (and enhancements) are presented in section 4. Finally, we summarize the paper in section 5.

2. ANALYSIS OF ANARCHY ONLINE GAME TRAFFIC

To look at an example, we have analyzed a one hour packet trace[1], obtained using tcpdump, containing all packets from one of a few hundred game regions in Funcom's popular MMORPG Anarchy Online. Each machine in the centralized server cluster manages one or more regions, i.e., no region spans several machines. Furthermore, Anarchy Online keeps point-to-point, default (New Reno) TCP connections open, from one or more Linux servers to every client. We found approximately 175 distinct connections in this packet dump, and knowing that the servers are located in the US, one can assume from the observed minimum latencies that there are players concurrently located in the US, Europe and Asia.

Some of our findings are depicted in figure 1. From figure 1(a), we see that the average RTT is somewhat above 250 ms with variations up to one second, i.e., these RTTs make the game playable [7]. However, looking at the maximum application delay (time for receiving a successful acknowledgment) which may include several retransmissions as a result of packet loss, we have extreme worst case delays of up 67 (!) seconds. Obviously, we cannot distinguish between lost packets and lost acknowledgments in this server-sided trace, but we can see the potential for several second-long delays in delivering packets to the application on the client side. Furthermore, figure 1(b) shows that, on average, less than one packet is sent per RTT. Combined with the measured RTTs, we see that the number of packets per second is low, below 4 packets per second. Considering that each packet is very small (about 120 bytes on average, see table 1), this demonstrates how thin the individual streams are. Finally, figure 1(c) shows that groups of these connections have a high probability of shared paths. It visualizes the results of a Wilcoxon test [11] of connection pairs that checks whether their RTT values stem from the same RTT base value set. A dot in the figure shows that two connections with high probability share a path.

With respect to losses, the original trace shows a loss probability of slightly less than 1 % but contains several instances of 6 retransmissions. This implies that it is not the loss rate itself that is unacceptable, but the occasional huge delays when multiple retransmissions are needed. Moreover, we have not found any correlation between losses of the various connections in the trace, and we would like to conclude that losses across the connections in the trace are uncorrelated. This would imply that losses are not due to server-sided bottlenecks. We use this as a working assumption, however, the thinness of the streams may actually hide loss correlation.

In summary, the lag that is experienced by players with lossy connections can be huge, and the thinness of the streams implies that retransmissions occur more or less only due to timeouts, since less than one packet is sent per RTT. Simultaneously, the congestion window grows very slowly even in slow start since several writes will be merged into a single packet, and repeated losses can keep the congestion window size close to 1. Thus, the main challenge for this kind of traffic is not the bandwidth, but the delay due to retransmissions that is typical for this kind of distributed multimedia application today. Therefore, it is important to note and deal with the most important observations, i.e., the connections are so thin that 1) they hardly ever trigger fast retransmissions but mainly retransmit due to timeout and 2) TCP's congestion control does not apply, i.e., the TCP stream does not back off. We therefore consider it reasonable to look at the TCP stack for an alleviation. Additionally, there are several connections that probably share several links in the path to the server, and we should use this observation to conserve network resources.

3. RELATED WORK

The body of work that has analyzed game traffic has grown considerably in the recent past. For example, Claypool [7] has investigated how latency affects the perceived quality for real-time strategy games where the results show that some latency is tolerable. Moreover, Feng et al. [5, 9] provide a comprehensive analysis of Counter-Strike traffic

[1]We have mostly looked at server to client communication as the client to server traffic is very smooth, and we do not have any control over the client TCP in order to make changes.

and investigate traces of several games concerning the predictability of game workloads. There, the conclusion is that game traffic varies strongly with time and with the attractiveness of the individual game. Chen et al. [6] investigate the traffic of MMORPGs. They find that streams are in general very thin, but that they are also bursty and show a correlation of inter-arrival times on the minute scale within individual streams. Furthermore, fitting multi-player game traffic to probability distributions is described by Borella [1].

Little work has been done on providing network-related mechanisms specifically for games, but Busse et.al. [3] present an admission control algorithm for a wireless network. They show its suitability for a simple game whose bandwidth usage approximately follows a Poisson distribution. However, we do not know any papers that analyze how underlying mechanisms affect the game performance and make proposals for improving the support for our kind of network traffic in the general Internet.

In Anarchy Online, TCP is used for communication, and we assume that this will remain the appropriate choice for MMORPGs for a while, due to the current state of firewall technology. We will therefore look more closely at TCP. Many variations of TCP exist to improve the throughput of individual streams without penalizing other TCP streams under various conditions.

The most basic TCP variant that is included in the Linux kernel is New Reno. It performs the usual slow start mechanism, halves the congestion window threshold and reduces the congestion window to 1 on timeout. If it receives duplicate acknowledgments it also performs fast retransmit and a variation of fast recovery that improves on TCP Reno. In Linux, it can be combined with selective acknowledgments (SACK). SACK was developed independently from New Reno. It resolves the ambiguity in senders' interpretation of duplicate ACKs by listing additional ranges of arrived bytes behind a gap. Also available are duplicate SACKs (DSACKs) that allow the receiver to inform that an ACK does not belong to new but to duplicate data. Forward acknowledgment (FACK) [12] is a SACK extension that keeps track of the amount of data that has been received and triggers a fast retransmit when the receiver indicates in a SACK block that it has a gap that is at least three segments long.

In addition to these variations, Linux offers a variety of TCP variations meant for higher throughput over connections with a high bandwidth-delay product. Binary increase congestion control (BIC) [14] tries to find the fair share of the bandwidth faster than Reno by performing a binary search for the number of packets per RTT that can be sent without packet loss. However, this happens only between two thresholds for the congestion window size. Below the lower threshold, normal congestion window development applies. Above the higher threshold, additive increase is applied. Vegas [2] uses RTT measurements to detect whether queue lengths increase, and stops to increase transmission speed when they do. This leads to very stable operation if only Vegas is used. The Linux implementation does not modify retransmissions in any way, but within one RTT, it can reaffirm the appropriate transmission speed after the connection has been temporarily idle. Westwood [4] avoids Vegas' problem of loosing bandwidth in competition with New Reno connections. It uses the flows of ACKs instead of RTTs to estimate the bandwidth. This algorithm does not modify basic loss recovery either.

There are, however, TCP variations not included in Linux that make such changes. A variation is fixed-RTO [8] that is meant to improve TCP performance over wireless networks. It does this, but on the general Internet it would obviously be considerably more aggressive in recovering after a congestion event than the other TCP variations.

4. EXPERIMENTS

Figure 1 shows that the basic average RTT between the Anarchy Online servers and the clients is rather high. But sporadically, clients experience really huge lag (delay in the application). This might be devastating to the perceived game experience. It is easily confirmed that the problem is due to the thinness of the streams. Few packets sent per RTT prevent fast retransmission, the congestion window does hardly grow, and packet loss is only detected by timeout, which additionally reduces the congestion window to 1 MSS. In addition, the sender reacts to repeated losses with exponential back-off.

To see whether the existing TCP variants in Linux could improve the situation, we replayed the packet trace through emulated lossy networks. FACK, for example, does not require triple duplicate ACKs to trigger retransmission and could reduce average lag, although probably not maximum. Additionally, we have looked at the possible gain of sending all the data in one single connection to save network resources. The idea is that the packets are redistributed to the clients by a proxy server.

4.1 Test Setup

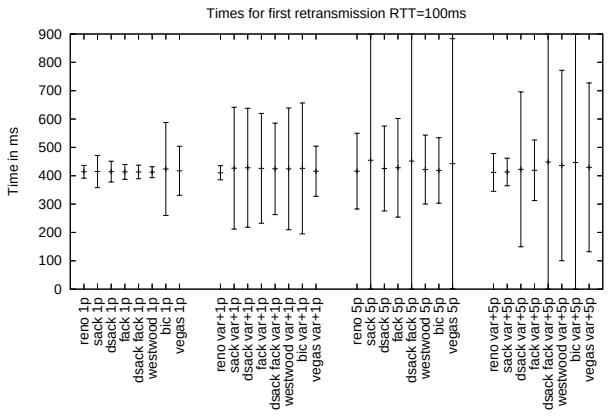
We performed the tests using two small local networks and three Pentium4 machines. The server application regenerates and sends all the original packets in the trace to the client (retransmissions present in the trace are not sent). To simulate delay, jitter and loss, we used *netem* (Linux 2.6.15 kernel) to add 100 ms and 300 ms delay with and without 10 % jitter. Due to instabilities of netem, we used a separate packet dropper implemented on a network processor (IXP2400) to drop 1 % and 5 % of the packets, respectively. The properties apply both ways in the network, i.e., since we have as many lost acknowledgments as lost packets, the client will have received about half of the retransmitted packets early, improving the actual situation from the perspective of the gamer.

4.2 Replaying the Game using Different Linux TCP Variants

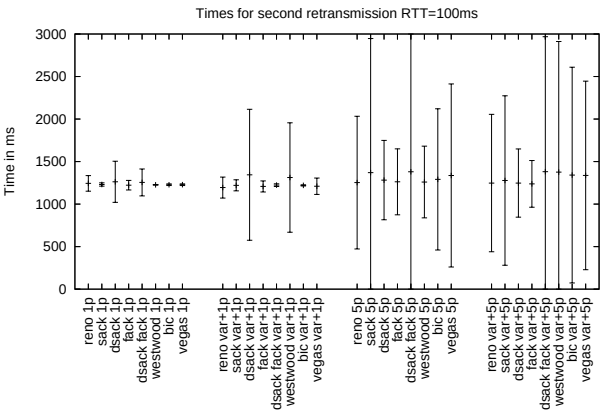
To see if there is any difference in how the various Linux TCP versions perform with respect to retransmission delay when low-rate, real-time event streams are sent to clients, we tested the existing Linux variants, i.e., New Reno (plain, with SACK, DSACK, FACK, and with DSACK and FACK), Westwood, BIC and Vegas. Additionally, we ran the tests with and without LOWLAT and FRTO turned on, but these options did not change anything.

Figure 2[2] shows the average time between the first transmission of a packet from a server and the retransmission that is finally successfully acknowledged. Each of the subfigures shows the average time for the 8 TCP settings in

[2]The results in figure 2 and 3 show the 100 ms delay only. However, the 300 ms experiments show identical results only with higher delays according to a higher RTT.



(a) Successful 1st retransmission



(b) Successful 2nd retransmissions

Figure 2: Average retransmission delay, simplex streams, 100ms delay

the 2.6.11 kernel in various situation. The figures are arranged in four blocks according to network properties, i.e., 1 % loss/no delay variation, 1 % loss/10 % delay variation, 5 % loss/no delay variation and 5 % loss/10 % delay variation.

Looking at the results, there are only small variations between the different options available in Linux. The first retransmissions are delayed by more or less the same amount of time[3] (about 400 ms, figure 2(a)). Since nearly all retransmissions are triggered by timeouts, the second (and later) retransmission have delays according the TCP backoff mechanism, i.e., about 1200 ms as shown in figure 2(b) for the second retransmission.

Obviously, changing the TCP version itself has no or very little effect in this scenario. Therefore, to see if we can improve the gaming quality by other means, we have experimented with a modified server architecture introducing proxies and a modified retransmission timeout (RTO) calculation at the sender in section 4.3 and 4.4, respectively.

4.3 Modified Architecture

To see the effect on delay and server resources, we compared individual connections to every client with a single connection carrying the multiplexed streams for all clients to an imaginary proxy which performs operations on behalf of the server and splits the single multiplexed TCP connection to each individual client. The test depicted in figure 1 has shown that quite a few clients probably share a path, so the proxy approach makes sense in that manner. A separate consideration is the increase in end-to-end latency that is naturally introduced by a proxy architecture. Minimal and average end-to-end latency would degrade, and the increase in bandwidth share that comes with TCP connection splitting is irrelevant in our scenario. However, we would benefit from improved loss recovery times.

Several interesting observations can be made. First, since packets are small (far below 100 bytes), the appended headers comprise a huge part of the number of transmitted bytes. As table 1 shows (for FACK), a lot of resources can be saved at the server. In our scenario, where about 175 individual thin streams are multiplexed into a single connection with an average data rate of 0.122 Mbps, we have reduced the amount of data and the number of packets by approximately 40 %. While this may seem irrelevant for the individual connection, an operator of a successful MMORPG has to dimension the access network for tens or even hundreds of thousands of concurrent connections.

Besides this, we see also the hoped-for improvement of per-client latency. We used the settings from section 4.2 to emulate multiplexed streams (figure 3). Comparing the two alternatives, we see that the average latency experienced over multiplexed connections is lower if the packet loss rate is low. In this approach, fast retransmissions are typical. This implies that the multiplexed stream is subject to congestion control, and an MMORPG that uses this approach would have to be able to scale the multiplexed stream to adapt to the available bandwidth. It is also important to note that the reduction in retransmission times is less pronounced when the latency is higher.

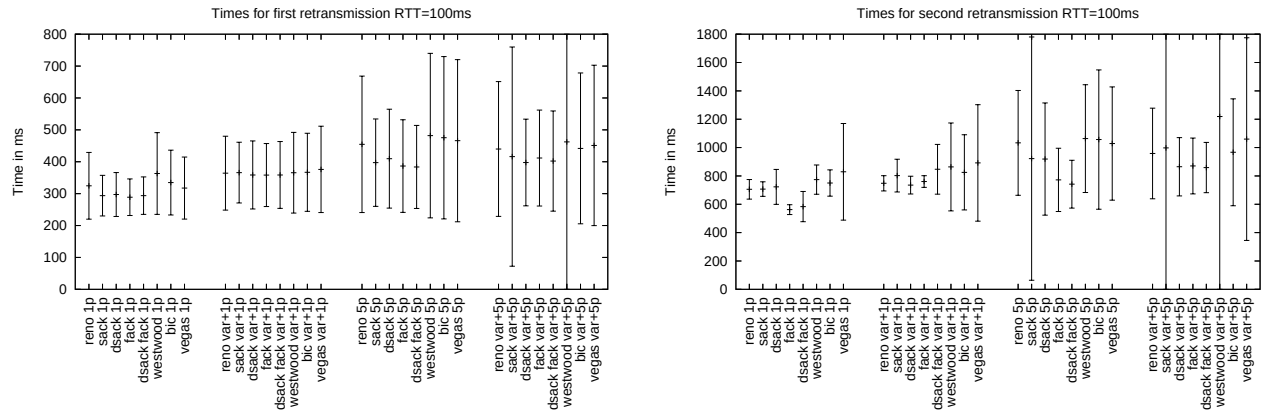
The high-speed variations do not perform well in the thin streams scenario. We found this particularly surprising for Westwood+, which is meant for efficient transfer over wireless connections. However, the implementation does never touch the smoothed RTT which is used for the RTO calculation; Westwood+ is meant to work well in the wireless scenario by not simply halving the congestion window in case of loss. If throughput is not the issue, it provides no advantage. Similarly, Vegas affects only congestion control.

According to figure 3, the most promising of all TCP settings in our scenario are the more recent extensions to New Reno, DSACK and/or FACK. For FACK, figure 4 shows a direct comparison of average latencies between direct and multiplexed connections. It demonstrates that the average latency is lower for the multiplexed connections, but it shows also that the advantage is lost when the packet loss rate is very high. In the following section, we are therefore looking at a the potential of a protocol variation.

[3]The timeout calculation in Linux increases the average delay with increasing RTT variance. However, with a maximum 10 % variance, this variation is too small to have a visible impact.

	total time	total #bytes	total #packets	average packet size	bandwidth requirement
simplex	1855 s	44974424 bytes	372711	120 bytes	0.194 Mbps
multiplex	1804 s	27582922 bytes	113149	243 bytes	0.122 Mbps

Table 1: Packet statistics (New Reno with FACK, 100 ms delay, 1% loss)



(a) Successful 1st retransmission

(b) Successful 2nd retransmissions

Figure 3: Average retransmission delay, a multiplexed stream, 100 ms delay

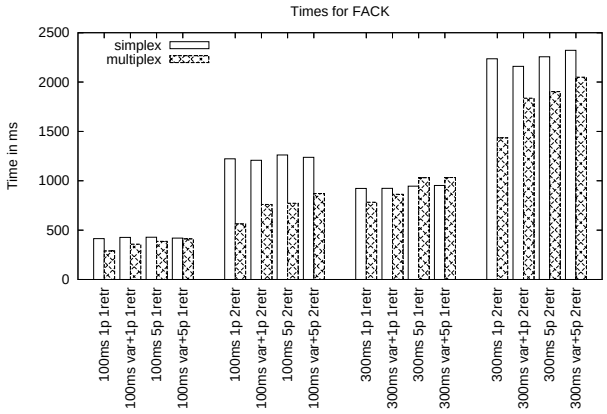


Figure 4: Average retransmission delay, comparing FACK results

4.4 Modified Kernel

To see whether we can gain anything from making the TCP retransmission scheme slightly more aggressive to better support thin real-time event streams without violating the basic congestion mechanisms, we modified a 2.6.15 kernel. The default standard RTO calculation and the traditional back-off mechanism is implemented as (using Linux notation[4])

$$min([(srtt \gg 3) + rttvar] \ll n, TCP\_RTO\_MAX)$$

where TCP_RTO_MAX is 2 minutes (default), $srtt$ is the smoothed RTT (7/8 srtt + 1/8 new rtt), $rttvar$ is the variance of the RTT (3/4 medium rtt deviation + 1/4 new rtt) and n is the number of retransmissions. In our modified scheme, we used a new RTO calculation if less than 3 packets are "in-flight" ($packets\_out < 3$), i.e., when we assume a thin stream that will not cause congestion. This RTO calculation is implemented as

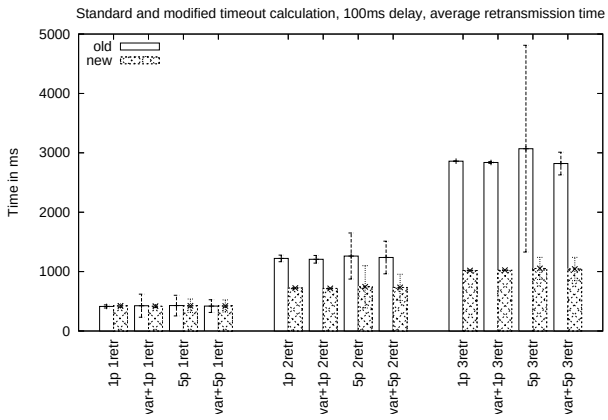
$$min([(srtt \gg 3) + min((srtt \gg 4), rttvar)], TCP\_RTO\_MAX).$$

Thus, as long as the traditionally calculated timeout is not shorter, this scheme uses the smoothed RTT only for the RTO calculation and does not apply back-off. Note, however, that our experimental modification is not meant to provide a perfect formula for thin streams. For example, the approach is not well suited for very high loss rates or correlated losses, and aggressive piggy-backing on other packets should also be considered. However, this is subject for further research, and we wanted here only to see if such a scheme could be used without wasting too many resources in additional retransmissions.

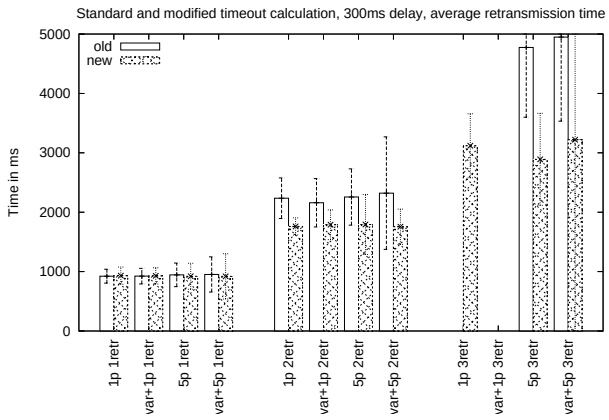
To see how the modified RTO scheme influences the retransmission delay, we reran a selection of the tests using New Reno and the basic extensions. The average retransmission delay with standard deviation for FACK is shown in figure 5 (the other results are similar). The retransmission latency can never be higher than in the original scheme, but the improvement increases considerably with the number of retransmissions that are necessary, i.e., the lag experienced by the player due to repeated packet loss is reduced.

On the other hand, the approach ignores $rttvar$ and tends to choose a more aggressive retransmission timer setting. This may increase the number of unnecessary retransmissions due to early timeouts. In table 2, we present the num-

[4]Other implementations may use other formulas like $RTO = rtt + 4 \times mdev$ which for example is used in BSD.



(a) 100 ms path delay, successful 1., 2. and 3. retransmissions



(b) 300 ms path delay, successful 1., 2. and 3. retransmissions

Figure 5: Comparing standard and modified RTO, New Reno with FACK

			delay	total	max	1st retrans			2nd retrans			3rd retrans		
	loss	delay	variance	#packets	#retrans.	total	self	incl.	total	self	incl.	total	self	incl.
Standard	1%	100ms	0%	184230	3	4528	3400	1128	65	53	12	2	2	0
	1%	100ms	10%	184213	3	4319	3298	1030	69	60	9	3	2	1
	1%	300ms	0%	182687	3	4282	2737	1545	32	22	10	0	0	0
	1%	300ms	10%	182855	3	4318	2717	1601	43	30	13	0	0	0
	5%	100ms	0%	182480	4	20532	15187	5345	1236	1055	181	127	109	18
	5%	100ms	10%	182820	4	20557	15253	5304	1153	945	208	115	96	19
	5%	300ms	0%	176573	5	18798	11729	7069	855	629	226	54	45	9
	5%	300ms	10%	175851	4	19290	12081	7209	945	705	240	93	65	28
Modified	1%	100ms	0%	184164	3	4642	3369	1273	58	49	9	2	2	0
	1%	100ms	10%	184145	3	4680	3396	1284	65	51	14	1	1	0
	1%	300ms	0%	179416	3	4239	2709	1530	43	32	11	4	2	0
	1%	300ms	10%	179927	3	4203	2659	1544	46	35	11	0	0	0
	5%	100ms	0%	177401	4	19115	14654	4461	1433	1185	248	143	121	22
	5%	100ms	10%	182320	6	19800	15158	4642	1352	1135	217	116	101	15
	5%	300ms	0%	161321	4	16882	11527	5355	1119	846	273	93	76	17
	5%	300ms	10%	159906	4	16708	11413	5295	1111	838	273	98	78	20

Table 2: Statistics for retransmissions using FACK

ber of experienced retransmissions, again represented by the FACK results. Here, the "self" column contains the number of retransmissions of a particular packet, while the "incl" column provides the number of retransmissions that are piggy-backed in (re)transmissions of another packet. Nevertheless, comparing the numbers with respect to the total number of transmitted packets, the two schemes are almost identical. For example, the new scheme is 0.07% higher for 100 ms delay/1% drop, but 0.5% lower if we increase to 5% drop – both for the first retransmission.

With respect to delivering time-dependent packets, our results show that it should be possible to implement an improved scheme for thin real-time event streams that improves retransmission delay, gives approximately the same number of retransmissions and does not apply to other situations, whereas for example fixed RTO [8] would also retransmit quickly when the RTT in a high bandwidth situation varies strongly.

5. CONCLUSION

Most existing work on TCP is aimed at providing the highest possible throughput to individual TCP connections while not violating the fairness principle. This is also the aim of the more recent options in the Linux kernel. At the same time, researchers discuss more frequently the use of TCP for time-dependent multimedia applications that do not always exhaust the bandwidth share that is available to their connection.

This is particularly true for the very thin game traffic streams examined in this paper. This scenario is hardly investigated and badly supported in TCP. For the individual stream, this lack results in a harsh penalty for not exhausting its bandwidth share. On the other hand, these streams appear in large numbers without any means of reacting to congestion.

Here, we have looked at the first issue and shown two approaches that can reduce the lag experienced at the application layer. Both the multiplexing of streams into a single

```
192.168.12.2:39106 - 192.168.2.50:12000
  num         time/sec
  126     PKT t 463.354 seq 5546 len  37    ;; packet 126 (seq 5546) is sent
  127     ACK t 463.681 ack 5546            ;; acknowledgement, receiver expects seq 5546
  128     PKT t 463.691 seq 5583 len  70    ;; packet 128 (seq 5583) is sent
  129     ACK t 463.933 ack 5583            ;; acknowledgement, receiver expects seq 5583
  130     PKT t 464.199 seq 5653 len  83    ;; packet 130 (seq 5653) is sent
  131     PKT t 464.726 seq 5736 len  94    ;; packet 131 (seq 5736) is sent
  132 RTR PKT t 464.737 seq 5583 len 153    ;; retransmission of packet 128, unacknowledged packet 130 is piggybacked
  133 DUP ACK t 464.845 ack 5583            ;; duplicate acknowledgement, receiver expects seq 5583
  134     PKT t 465.019 seq 5830 len  25    ;; packet 134 (seq 5830) is sent
  135 DUP ACK t 465.329 ack 5583            ;; (second) duplicate acknowledgement, receiver expects seq 5583
  136     PKT t 465.329 seq 5855 len 228    ;; packet 136 (seq 5855) is sent
  137     ACK t 465.361 ack 5830            ;; acknowledgement, receiver expects seq 5830
```

Figure 6: A retransmission

TCP connection and a more aggressive timeout retransmission approach promise a reduced lag. We do therefore intend to pursue further options for improvement. For example, we see that the use of TCP has a disadvantage over the use of UDP. A correctly received TCP packet following a lost packet is blocked from delivery to the application until the lost packet is eventually retransmitted successfully. The thin streams would also permit an alternative handling. For example, an application of the piggy-backing approach seen in figure 6, in which the unacknowledged packet 130 is piggy-backed with the retransmission of packet 128, could also be applied to first time transmissions if the a packet is not acknowledged by the simple RTO. A new packet would include the payload of the previous packet as well, e.g., piggy-backing packet number 126 in packet 128, doubling its chances of being delivered just as quickly as a UDP packet.

6. ACKNOWLEDGEMENTS

We would like to thank Funcom, Jarl Christian Berentsen in particular, for information about Anarchy Online and the trace used in this study.

7. REFERENCES

[1] Borella, M. S. Source models of network game traffic. *Elsevier Computer Communications 23*, 4 (Feb. 2000), 403–410.

[2] Brakmo, L. S., O'Malley, S. W., and Peterson, L. L. TCP Vegas: new techniques for congestion detection and avoidance. In *Proceedings of the ACM International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)* (London, UK, 1994), ACM Press, pp. 24–35.

[3] Busse, M., Lamparter, B., Mauve, M., and Effelsberg, W. Lightweight QoS-support for networked mobile gaming. In *Proceedings of the Workshop on Network and System Support for Games (NETGAMES)* (Portland, OR, USA, 2004), pp. 85–92.

[4] Casetti, C., Gerla, M., Mascolo, S., Sanadidi, M. Y., and Wang, R. TCP Westwood: end-to-end congestion control for wired/wireless networks. *Wireless Network 8*, 5 (2002), 467–479.

[5] Chambers, C., Wu-chang Feng, Sahu, S., and Saha, D. Measurement-based characterization of a collection of on-line games. In *Proceedings of the USENIX Internet Measurement Conference (IMC)* (Berkeley, CA, USA, 2005), pp. 1–14.

[6] Chen, K.-T., Huang, P., Huang, C.-Y., and Lei, C.-L. Games traffic analysis: An MMORPG perspective. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)* (Stevenson, WA, USA, 2005), ACM Press, pp. 19–24.

[7] Claypool, M. The effect of latency on user performance in real-time strategy games. *Elsevier Computer Networks 49*, 1 (Sept. 2005), 52–70.

[8] Dyer, T. D., and Boppana, R. V. A comparison of TCP performance over three routing protocols for mobile ad hoc networks. In *Proceedings of the International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc)* (Long Beach, CA, USA, 2001), ACM Press, pp. 56–66.

[9] Feng, W.-c., Chang, F., Feng, W.-c., and Walpole, J. Provisioning on-line games: a traffic analysis of a busy Counter-strike server. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement* (Marseille, France, 2002), pp. 151–156.

[10] Funcom. Anarchy online. http://www.anarchy-online.com/, Feb. 2006.

[11] Larsen, R. J., and Marx, M. L. *An Introduction to Mathemetical Statistics and Its Applications*. Prentice Hall, 1986.

[12] Mathis, M., and Mahdavi, J. Forward acknowledgement: refining TCP congestion control. In *Proceedings of the ACM International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)* (Palo Alto, CA, USA, 1996), ACM Press, pp. 281–291.

[13] Whang, L. S.-M., and Kim, J. Y. The online game world as a product and the behavioral characteristics of online game consumers as role player. In *Proceedings of the Digital Games Research Association International Conference (DIGRA)* (Vancouver, Canada, June 2005).

[14] Xu, L., Harfoush, K., and Rhee, I. Binary increase congestion control for fast long-distance networks. In *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)* (Hong Kong, China, 2004).